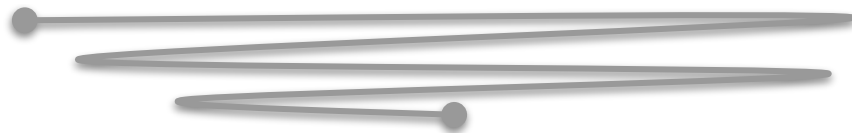


# Laboratorio de Datos



## Introducción a Python // Parte 02



2do. Cuatrimestre  
2023



DEPARTAMENTO  
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA



# Laboratorio de Datos

## Introducción a Python - Parte 02

... por Manuela Cerdeiro (y modificaciones mínimas de P. Turjanski)

# Contenido

- + Repaso ejercicios
- + Python desde la terminal
- + Correr archivos .py
- + IDE - Spyder
- + Funciones
- + Diccionarios
- + Módulos
- + Manejo de archivos

# Repaso

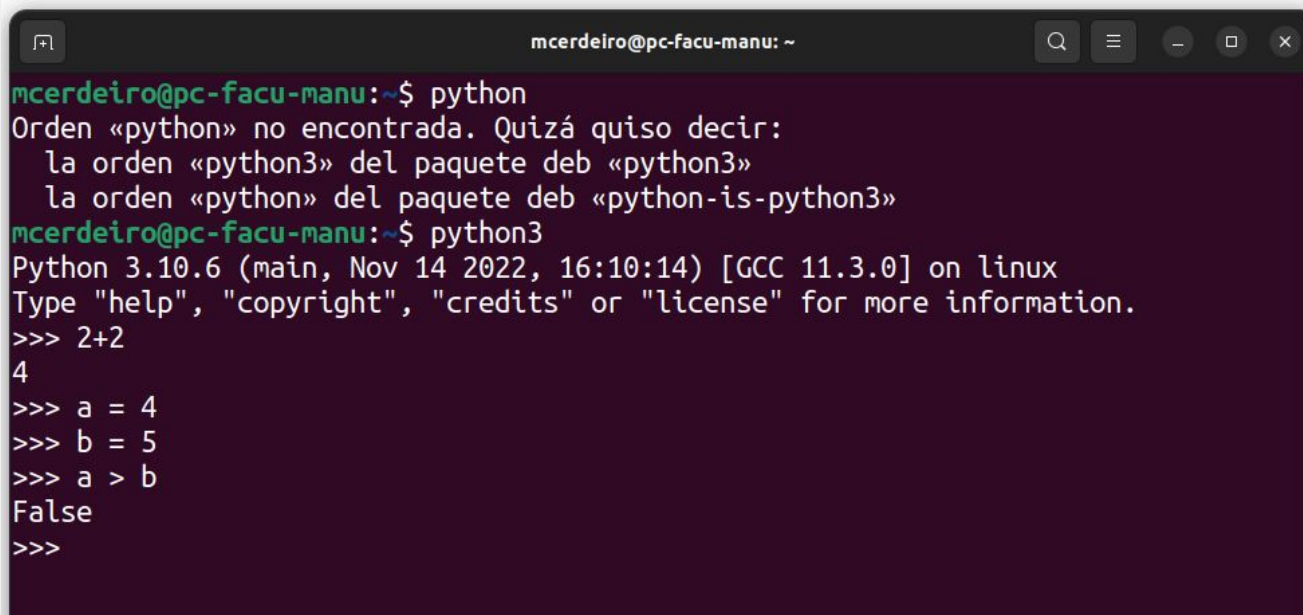
# Ejercicios

- + Geringoso
- + Rebotes
- + Inclusive

# Python desde la terminal

# Python desde la terminal

En una terminal, usamos "python" o "python3" para abrir un intérprete de python.

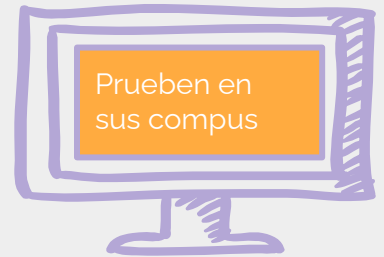
A terminal window with a dark background and light green text. The window title is "mcerdeiro@pc-facu-manu: ~". The user enters "python" and receives an error message: "Orden «python» no encontrada. Quizá quiso decir: la orden «python3» del paquete deb «python3» la orden «python» del paquete deb «python-is-python3»". Then the user enters "python3" and the Python 3.10.6 interpreter starts. The user enters "2+2" and gets "4", then "a = 4", "b = 5", "a > b" and gets "False".

```
mcerdeiro@pc-facu-manu:~$ python
Orden «python» no encontrada. Quizá quiso decir:
  la orden «python3» del paquete deb «python3»
  la orden «python» del paquete deb «python-is-python3»
mcerdeiro@pc-facu-manu:~$ python3
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> a = 4
>>> b = 5
>>> a > b
False
>>>
```

# Python desde la terminal

Al terminar, salimos con `exit()`

```
mcerdeiro@pc-facu-manu: ~$ python
Orden «python» no encontrada. Quizá quiso decir:
  la orden «python3» del paquete deb «python3»
  la orden «python» del paquete deb «python-is-python3»
mcerdeiro@pc-facu-manu:~$ python3
Python 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> a = 4
>>> b = 5
>>> a > b
False
>>> exit()
mcerdeiro@pc-facu-manu:~$
```





Archivos .py

# Uso de archivos .py

Podemos guardar el código como archivo de texto con extensión .py y luego ejecutarlo desde la terminal con python. Por ejemplo:

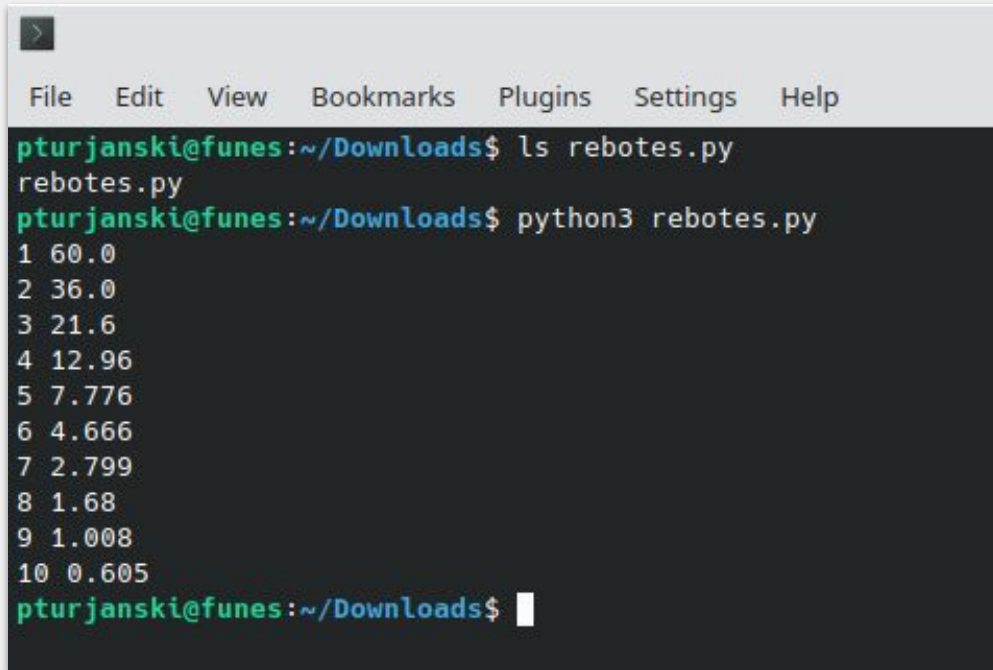
Creamos un archivo de texto, con este contenido, y lo guardamos como `rebotes.py`

```
# rebotes.py
```

```
altura = 100
i = 1
while i <= 10:
    # Calcula la nueva altura
    altura = 3/5*altura
    # Muestra en pantalla los nuevos datos
    print(i, end = ' ')
    print(round(altura,3))
    # Actualiza el contador de rebotes
    i+=1
```

# Uso de archivos .py

Ahora abrimos una terminal, nos situamos en el mismo directorio en el que está el archivo `rebotes.py` y ejecutamos `python3 rebotes.py`

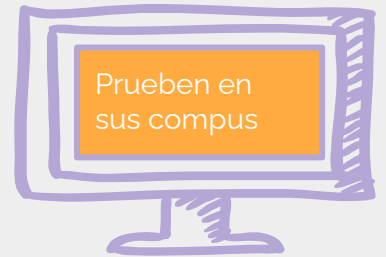
A screenshot of a terminal window with a light gray title bar and a dark background. The title bar contains a maximize button icon and a menu bar with the following items: File, Edit, View, Bookmarks, Plugins, Settings, and Help. The terminal shows the user 'pturjanski' at host 'funes' in the directory '~/Downloads'. The user enters the command 'ls rebotes.py', which outputs 'rebotes.py'. Then, the user enters 'python3 rebotes.py', which outputs a list of 10 numbers: 60.0, 36.0, 21.6, 12.96, 7.776, 4.666, 2.799, 1.68, 1.008, and 0.605. The prompt is ready for the next command.

```
>  
File Edit View Bookmarks Plugins Settings Help  
pturjanski@funes:~/Downloads$ ls rebotes.py  
rebotes.py  
pturjanski@funes:~/Downloads$ python3 rebotes.py  
1 60.0  
2 36.0  
3 21.6  
4 12.96  
5 7.776  
6 4.666  
7 2.799  
8 1.68  
9 1.008  
10 0.605  
pturjanski@funes:~/Downloads$
```

# Uso de archivos .py

También podemos ejecutar con `-i` y quedarnos en el intérprete de python luego de la ejecución del programa. Las variables quedan en el estado final del programa.

```
pturjanski@funes:~/Downloads$ python3 -i rebotes.py
1 60.0
2 36.0
3 21.6
4 12.96
5 7.776
6 4.666
7 2.799
8 1.68
9 1.008
10 0.605
>>> altura
0.6046617599999998
>>> i
11
>>> 
```



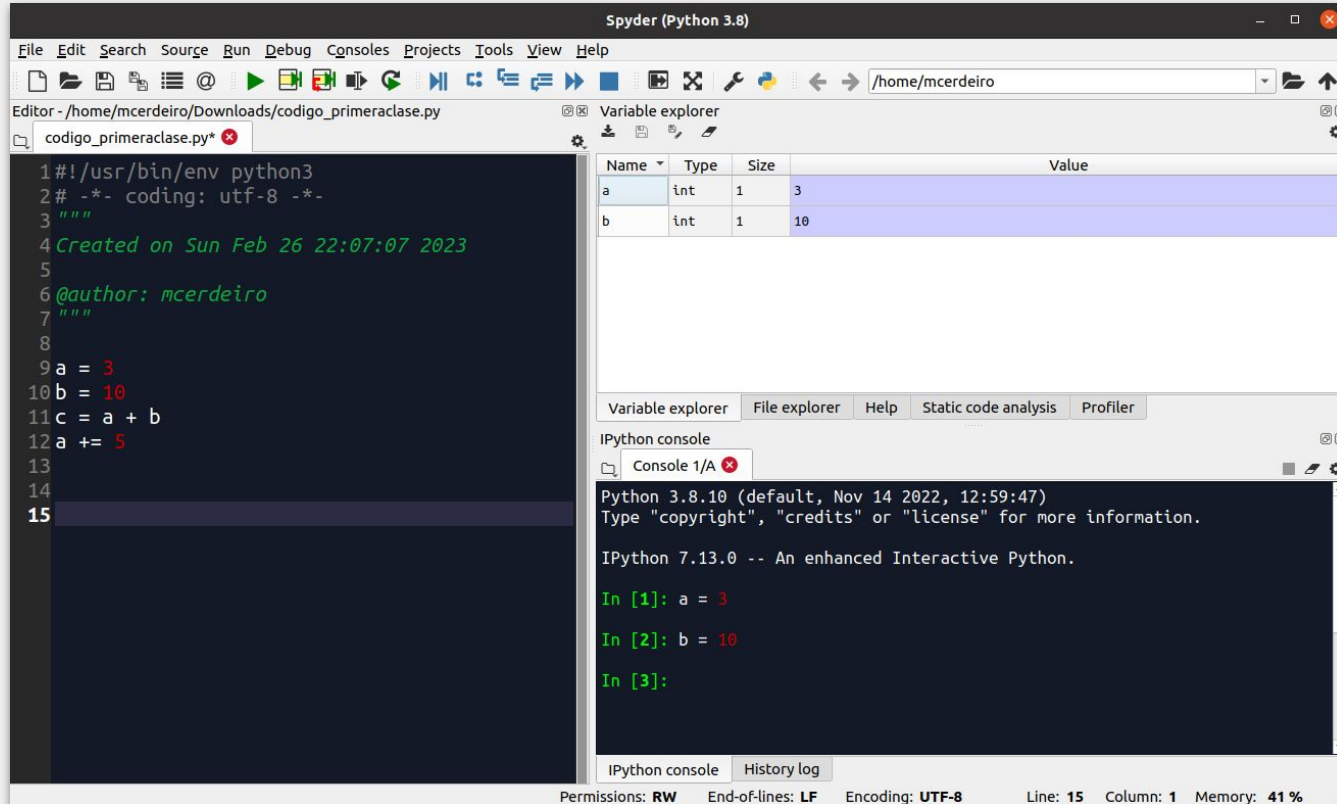
# Spyder



# Entorno de desarrollo integrado (*IDE*) - Spyder

- + Código abierto (open source)
- + Multiplataforma (sirve en linux, ios, windows...)
- + Es cómodo para escribir código, ejecutarlo, corregirlo, probarlo y utilizarlo, en el mismo entorno
- + El editor de texto remarca palabras clave del lenguaje
- + Tiene atajos (shortcuts) útiles (y modificables a gusto de cada unx)
- + Muy parecido a RStudio (Lenguaje de programación R)

# Entorno de desarrollo integrado (IDE) - Spyder



# Entorno de desarrollo integrado (IDE) - Spyder

The image shows the Spyder Python IDE interface with several components and annotations:

- Editor:** The main window for editing code. It contains a file named `codigo_primeraclase.py`. The code is as follows:

```
1#!/usr/bin/env python3
2# -*- coding: utf-8 -*-
3"""
4Created on Sun Feb 26 22:07:07 2023
5
6@author: mcerdeiro
7"""
8
9a = 3
10b = 10
11c = a + b
12a += 5
13
14
15
```

Annotation: **nombre del programa** (points to the filename `codigo_primeraclase.py` in the editor tab).
- Variable explorer:** A panel on the right showing the current state of variables in the program. It contains a table with the following data:

Name	Type	Size	Value
a	int	1	3
b	int	1	10

Annotation: **explorador de variables (muestra el estado del programa)** (points to the Variable explorer panel).
- IPython console:** A panel at the bottom right for running code. It shows the output of the code executed in the editor:

```
Python 3.8.10 (default, Nov 14 2022, 12:59:47)
Type "copyright", "credits" or "license" for more information.

IPython 7.13.0 -- An enhanced Interactive Python.

In [1]: a = 3
In [2]: b = 10
In [3]:
```

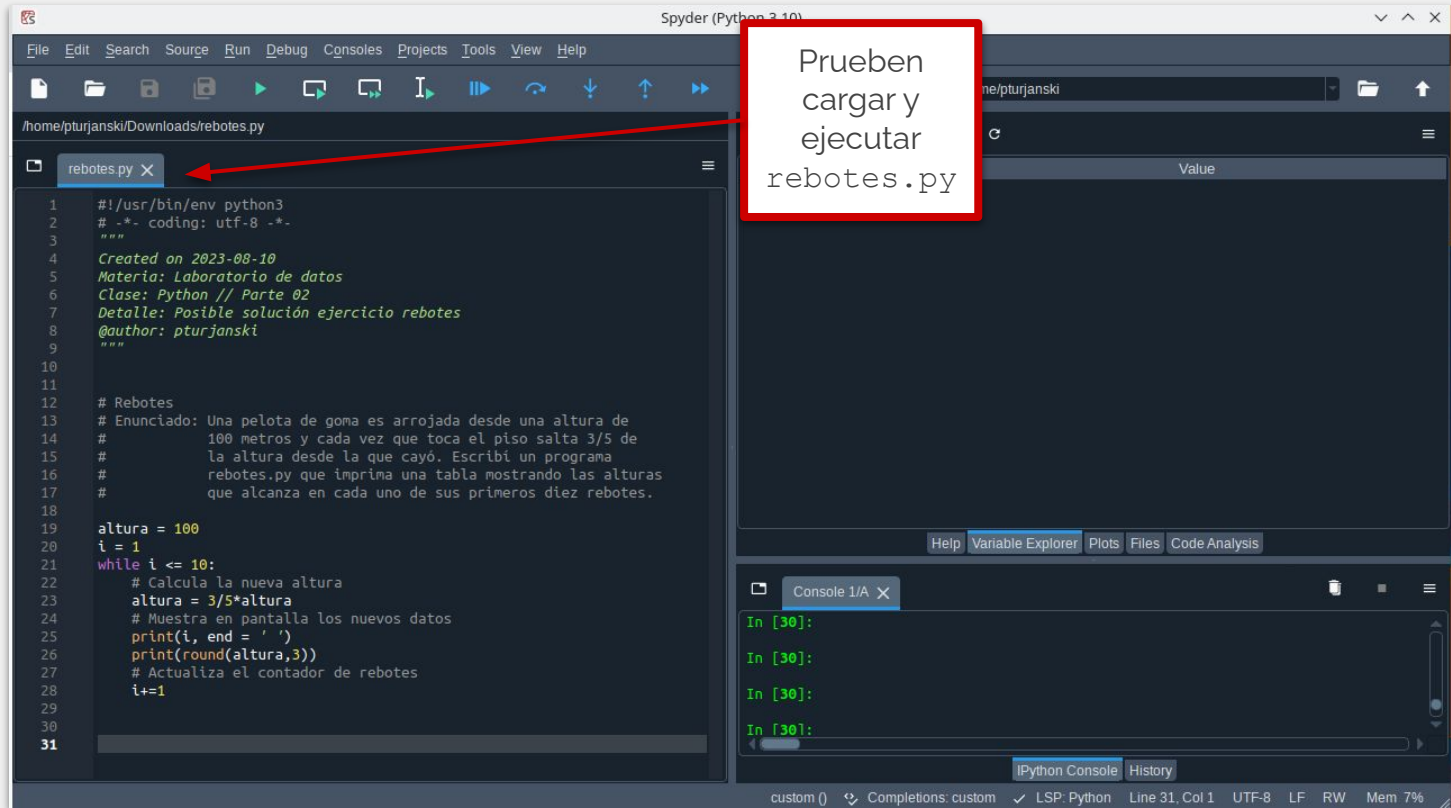
Annotation: **intérprete de python** (points to the IPython console panel).
- File explorer:** A panel at the top right showing the current directory and its contents. It displays `/home/mcerdeiro`.

Annotation: **directorio actual de trabajo** (points to the File explorer panel).
- Editor de texto:** The main area where the code is written and edited.

At the bottom of the window, the status bar shows: Permissions: RW, End-of-lines: LF, Encoding: UTF-8, Line: 15, Column: 1, Memory: 41 %.



# Entorno de desarrollo integrado (IDE) - Spyder



# Funciones

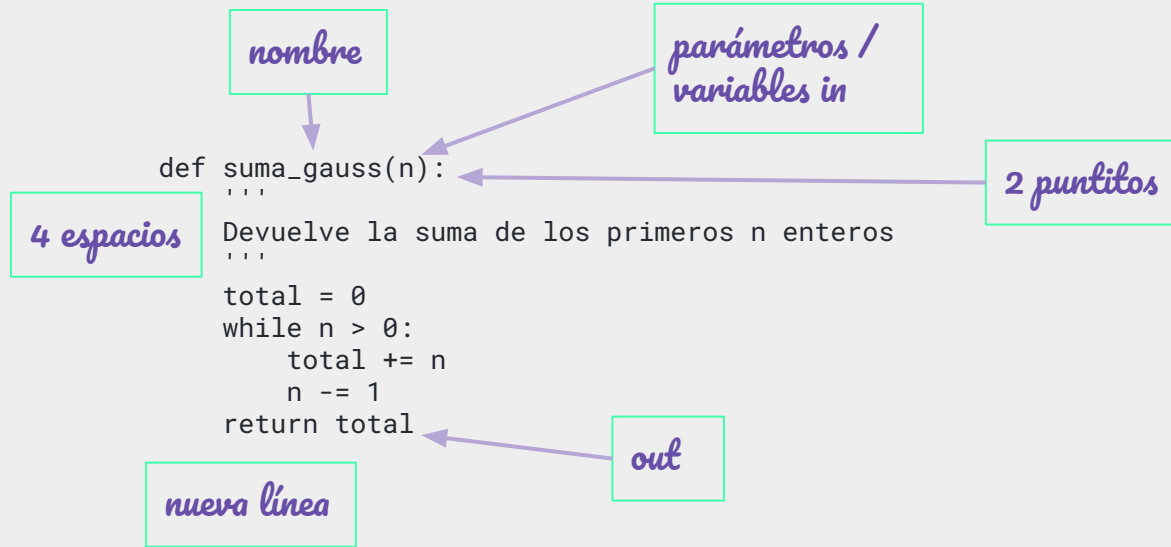
# Funciones

Las funciones son una herramienta para encapsular pedazos de código, facilitando su reutilización.

```
def suma_gauss(n):  
    '''  
    Devuelve la suma de los primeros n enteros  
    '''  
    total = 0  
    while n > 0:  
        total += n  
        n -= 1  
    return total
```

# Funciones

Definición de una función.



# Funciones

Definición de una función.

```
def suma_gauss(n):  
    '''  
    Devuelve la suma de los primeros n enteros  
    '''  
    total = 0  
    while n > 0:  
        total += n  
        n -= 1  
    return total
```

Llamada (uso) de una función.

```
suma_gauss(10)  
  
x = 25  
suma_gaus(x)
```

# Ejercicio

- ★ Definir una función **maximo(a,b)** que tome dos parámetros numéricos y devuelva el máximo



# Ejercicio

- ★ Definir una función **tachar\_pares(lista)** que tome una lista de números y devuelva una similar pero donde los números pares estén reemplazados por 'x'



# Funciones

A tener en cuenta

- ❑ Cuando una función involucra tipos de datos mutables, puede modificarlos (estén o no en el `return`).
- ❑ Repasar en las funciones de los ejercicios previos, ¿cuáles modifican variables y cuáles no?
- ❑ A veces esto es deseable y a veces NO. ¿Cómo lo podemos evitar?



# Copias - Probar en **pythontutor**

Las listas tienen un método para hacer copias.

```
a = [2,3,[100,101],4]
b = a.copy()
b == a # True
b is a # False
```

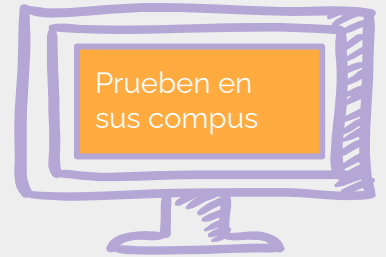
# Copias - Probar en **pythontutor**

```
a = [2,3,[100,101],4]
b = a.copy()
b == a # True
b is a # False
```

```
c = a
c == a # True
c is a # True
```

```
a.append(5)
```

```
print(b)
print(c)
```



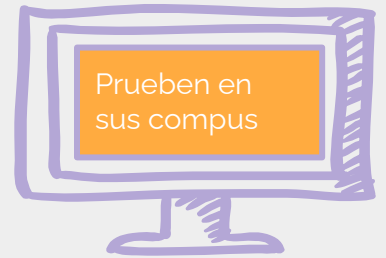
# Copias - Probar en **pythontutor**

```
a = [2,3,[100,101],4]
b = a.copy()
b == a # True
b is a # False
```

```
c = a
c == a # True
c is a # True
```

```
a[2].append(102)
```

```
print(b)
print(c)
```



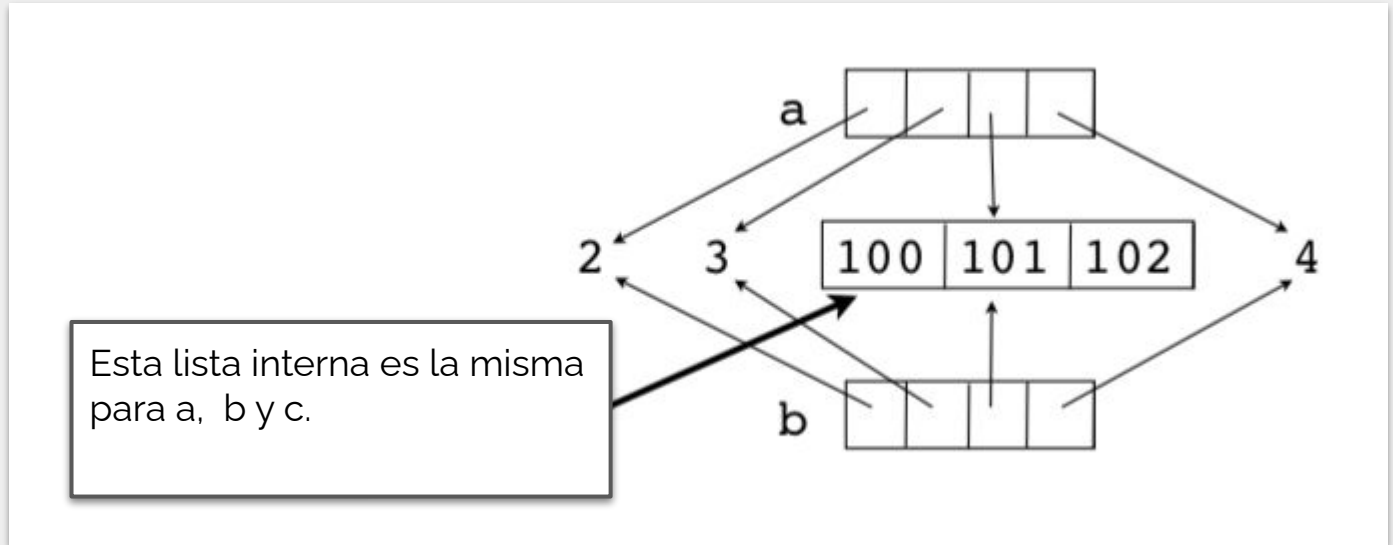
# Copias - Probar en **pythontutor**

```
a = [2,3,[100,101],4]
b = a.copy()
b == a # True
b is a # False
```

```
c = a
c == a # True
c is a # True
```

```
a[2].append(102)
```

```
print(b)
print(c)
```



# Deepcopy

```
import copy
```

```
a = [2,3,[100,101],4]
```

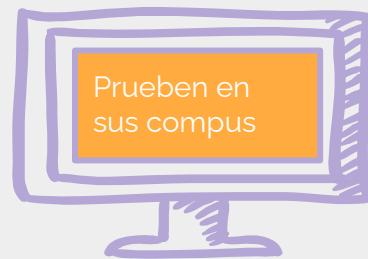
```
b = copy.deepcopy(a)
```

```
a.append(5)
```

```
print(b)
```

```
a[2].append(102)
```

```
print(b)
```



# Ejercicio

Probar las funciones anteriores utilizando y no utilizando copias.



# Diccionarios

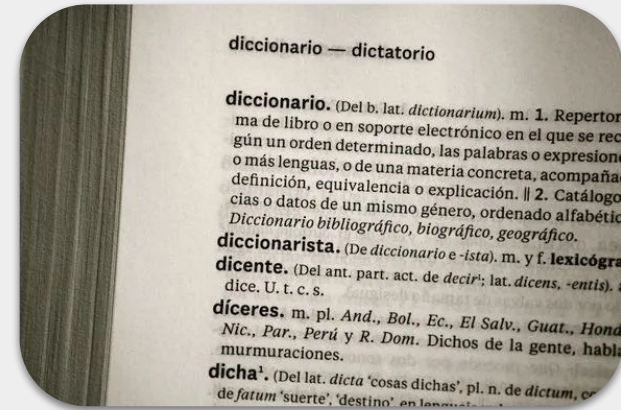
# Diccionarios

Los diccionarios son útiles si vamos a querer buscar rápidamente (por claves).

- Se construyen con llaves { }
- Cada entrada tiene una clave y un valor, separados por dos puntitos :
- Las entradas se separan con comas

```
{clave1: valor1, clave2: valor2, ... }
```

- Se acceden con corchetes indicando una clave
- Tanto las claves como los valores pueden ser de distintos tipos de objetos
- Las claves deben ser de tipo inmutable





# Ejemplo

```
dias_engl = {'lunes': 'monday', 'martes': 'tuesday', 'miércoles': 'wednesday', 'jueves':  
'thursday'}
```

```
>>> dias_engl['lunes']  
'monday'
```

```
>>> dias_engl['viernes']  
Traceback (most recent call last):
```

```
  File "<ipython-input-33-ee0fa133453b>", line 1, in <module>  
    dias_engl['viernes']
```

```
KeyError: 'viernes'
```

```
>>> dias_engl['viernes'] = 'friday'      # agrego la entrada  
>>> dias_engl['viernes']  
'friday'
```

# Ejemplo

También se pueden armar y modificar agregando entradas:

```
feriados = {} # Empezamos con un diccionario vacío
```

```
# Agregamos elementos
```

```
feriados[(1, 1)] = 'Año nuevo'
```

```
feriados[(1, 5)] = 'Día del trabajador'
```

```
feriados[(13, 9)] = 'Día del programador'
```

```
# Modifico una entrada
```

```
feriados[(13, 9)] = 'Día de la programadora'
```

# Diccionarios

También se pueden armar a partir de una lista de tuplas (clave, valor).

```
>>> tuplasDeCuadrados = [(1,1), (2,4), (3,9), (4,16)]
>>> cuadrados = dict(tuplasDeCuadrados)
>>> cuadrados[2]
4
```

La función **zip** genera tuplas a partir de dos listas:

```
>>> basesCuadrado = [1,2,3,4]
>>> resulCuadrado = [1,4,9,16]
>>> cuadrados = dict(zip(basesCuadrado, resulCuadrado))
```

# Ejercicio

Construí una función `traductor_geringoso(lista)` que, a partir de una lista de palabras, devuelva un diccionario geringoso.

Las claves del diccionario deben ser las palabras de la lista y los valores deben ser sus traducciones al geringoso.

Por ejemplo:

```
>>> lista = ['banana', 'manzana', 'mandarina']  
>>> traductor_geringoso(lista)
```

```
{'banana': 'bapanapanapa', 'manzana': 'mapanzapanapa', 'mandarina': 'mapandaparipinapa'}
```



# Módulos

# Módulos

Si bien Python tiene muchas funciones que se pueden usar directamente, hay muchas otras que están disponibles dentro de módulos.

Un **módulo** es una **colección de funciones** que alguien (o una comunidad) desarrollaron y empaquetaron para que estén disponibles para todo el mundo.

Para que las funciones estén disponibles para ser utilizadas en mi programa, tengo que usar la instrucción **import**.

# Módulos

Si quiero generar números aleatorios, que están en el módulo random, tengo que escribir:

```
import random
prueba = random.random()
print(prueba)
prueba = random.random()
print(prueba)
prueba = random.random()
print(prueba)
```

```
random.seed(COMPLETAR con un NÚMERO)
prueba = random.random()
print(prueba)
prueba = random.random()
print(prueba)
```

Repetir varias veces ... ¿qué ocurre al utilizar la misma semilla (seed)?

# Módulos

Módulo math tiene funciones matemáticas.

```
import math
```

```
math.sqrt(16)
```

```
math.exp(x)
```

```
math.cos(x)
```

```
math.gcd(15, 12)
```



# Ejercicio

Escribir una función `general_tirar()` que simule una tirada de dados para el juego de la generala. Es decir, debe devolver una lista aleatoria de 5 valores de dados . Por ejemplo, si sale `2,1,1,2,2` debe imprimir `[2,1,1,2,2]`

Opcional.

Agregar al ejercicio `general_tirar()` que además imprima en pantalla si salió poker, full, generala, escalera o ninguna de las anteriores. Por ejemplo, si sale `2,1,1,2,2` debe devolver `[2,1,1,2,2]` e imprimir en pantalla `Full`



# Archivos

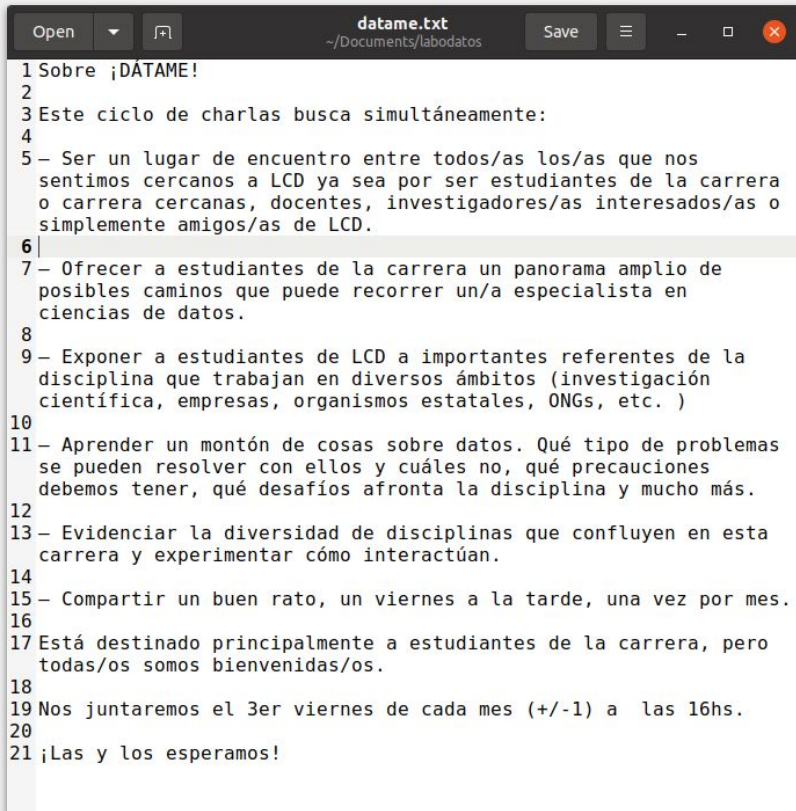
# Archivos

Frecuentemente vamos a utilizar una fuente de datos, que en muchos casos va a estar en un archivo.

Tenemos que poder manejar archivos: leer, crear, modificar, guardar archivos de distintos tipos.

```
f = open(nombre_archivo, 'rt' )      # abrir para lectura ('r' de read, 't' de text)
data = f.read()
f.close()
data
print(data)
```

# Ejemplo



```
1 Sobre ¡DATAME!  
2  
3 Este ciclo de charlas busca simultáneamente:  
4  
5 - Ser un lugar de encuentro entre todos/as los/as que nos  
   sentimos cercanos a LCD ya sea por ser estudiantes de la carrera  
   o carrera cercanas, docentes, investigadores/as interesados/as o  
   simplemente amigos/as de LCD.  
6  
7 - Ofrecer a estudiantes de la carrera un panorama amplio de  
   posibles caminos que puede recorrer un/a especialista en  
   ciencias de datos.  
8  
9 - Exponer a estudiantes de LCD a importantes referentes de la  
   disciplina que trabajan en diversos ámbitos (investigación  
   científica, empresas, organismos estatales, ONGs, etc. )  
10  
11 - Aprender un montón de cosas sobre datos. Qué tipo de problemas  
   se pueden resolver con ellos y cuáles no, qué precauciones  
   debemos tener, qué desafíos afronta la disciplina y mucho más.  
12  
13 - Evidenciar la diversidad de disciplinas que confluyen en esta  
   carrera y experimentar cómo interactúan.  
14  
15 - Compartir un buen rato, un viernes a la tarde, una vez por mes.  
16  
17 Está destinado principalmente a estudiantes de la carrera, pero  
   todas/os somos bienvenidas/os.  
18  
19 Nos juntaremos el 3er viernes de cada mes (+/-1) a las 16hs.  
20  
21 ¡Las y los esperamos!
```

Ejemplo:  
`nombre_archivo = 'datame.txt'`

```
nombre_archivo = 'datame.txt'  
f = open(nombre_archivo, 'rt' )  
data = f.read()  
f.close()
```

```
data  
print(data)
```

Python tiene dos modos de salida.  
En el primero, escribimos `data` en el intérprete y Python muestra la representación **cruda** de la cadena, incluyendo comillas y códigos de escape (`\n`).  
Cuando escribimos `print(data)`, en cambio, se imprime la salida **formateada** de la cadena.

# Archivos

```
with open(nombre_archivo, 'rt') as f:  # otra forma de abrir archivos
    data = f.read()
    # 'data' es una cadena con todo el texto en el archivo

data
print(data)
```

Para leer una archivo línea por línea, usá  
un ciclo for como éste:

```
with open(nombre_archivo, 'rt') as f:
    for l in f:
        # Procesar la línea
        print("->",l)
```

# Ejercicio

Escribir un programa que recorra las líneas del archivo `'datame.txt'` e imprima solamente las líneas que contienen la palabra `'estudiante'`.



# Archivos .csv

csv = comma separated values

- son “planillas” guardadas como texto en un archivo
- cada línea de texto es una fila de la planilla
- las comas separan las columnas

```
UAI Urquiza,19,48  
Rosario Central,19,43  
Platense,19,35  
Excursionistas,18,23
```

UAI Urquiza	19	48
Rosario Central	19	43
Platense	19	35
Excursionistas	18	23

# Archivos .csv

csv = comma separated values

Ejemplo:

nombre\_archivo = 'cronograma\_sugerido.csv'

Cuatrimestre	Asignatura	Correlatividad de Asignaturas
3	Álgebra I	CBC
3	Algoritmos y Estructuras de Datos I	CBC
4	Análisis I	CBC
4	Electiva de Introducción a las Ciencias Naturales	CBC
5	Análisis II	Análisis I
5	Álgebra Lineal Computacional	Álgebra I – Algoritmos y Estructuras de Datos I
5	Laboratorio de Datos	Algoritmos y Estructuras de Datos I
6	Análisis Avanzado	Análisis II, Álgebra I
6	Algoritmos y Estructuras de Datos II	Algoritmos y Estructuras de Datos I
7	Probabilidad	Análisis Avanzado
7	Algoritmos y Estructura de Datos III	Algoritmos y Estructuras de Datos II
8	Intr. a la Estadística y Ciencia de Datos	Lab de Datos, Probabilidad, Álgebra Lineal Computacional
8	Intr. a la Investigación Operativa y Optimización	Alg y Estruct de Datos III, Análisis II, Álgebra Lineal Computacional
8	Intr. al Modelado Continuo.	Análisis Avanzado, Álgebra Lineal Computacional, Alg y Estructura de Datos II



# Ejercicio

A partir del archivo `cronograma_sugerido.csv` armar una lista con todas las asignaturas del cronograma



# Archivos .csv

```
nombre_archivo = 'cronograma_sugerido.csv'
asignaturas = []

with open(nombre_archivo,'rt') as file:
    for line in file:
        datos_linea = line.split(',')
        asignaturas.append(datos_linea[1])

asignaturas.remove('Asignatura')
print(asignaturas)
```

# Ejercicio

Definir una función "cuantas\_materias(n) " que, dado un número de cuatrimestre (n entre 3 y 8), devuelva la cantidad de materias a cursar en ese cuatrimestre

Ejemplo:

```
>>> cuantas_materias(5)
```

```
3
```

```
>>> cuantas_materias(7)
```

```
2
```



# Módulo csv

Es útil para trabajar con archivos .csv

```
import csv
```

```
f = open(nombre_archivo)
filas = csv.reader(f)
for fila in filas:
    instrucciones
```

Acá **filas** es un iterador.

```
f.close()
```

Si la primera fila son encabezados, podemos leerlo así:

```
f = open(nombre_archivo)
filas = csv.reader(f)
encabezado = next(filas) # un paso del iterador
for fila in filas:        # ahora el iterador sigue desde la segunda fila
    instrucciones

f.close()
```

# Ejemplo

Definimos `registros(nombre_archivo)` que recorre el archivo indicado, conteniendo por ejemplo la información de un cronograma sugerido de cursada, y devuelve la información como una lista de diccionarios. Las claves de los diccionarios son las columnas del archivo, y los valores son las entradas de cada fila para esa columna.

```
def registros(nombre_archivo):
    lista = []
    with open(nombre_archivo, 'rt') as f:
        filas = csv.reader(f)
        encabezado = next(filas)
        for fila in filas:
            registro = dict(zip(encabezado, fila)) # Arma el diccionario de cada fila
            lista.append(registro)                # Agrega el diccionario a la lista
    return lista
```

# Ejercicio

Definir una función `materias_cuatrimestre(nombre_archivo, n)` que recorra el archivo indicado, conteniendo información de un cronograma sugerido de cursada, y devuelva una lista de diccionarios con la información de las materias sugeridas para cursar el n-ésimo cuatrimestre. Debe funcionar así:

```
materias_cuatrimestre('cronograma_sugerido.csv', 3):
```

```
[{'Cuatrimestre': '3',  
  'Asignatura': 'Álgebra I',  
  'Correlatividad de Asignaturas': 'CBC'},  
 {'Cuatrimestre': '3',  
  'Asignatura': 'Algoritmos y Estructuras de Datos I',  
  'Correlatividad de Asignaturas': 'CBC'}]
```



# TP - Arbolado Porteño [Grupal]

- Datos sobre árboles en CABA
- Enunciado -> Campus
- Resolución con Python
- Trabajar en grupos de 3
- No es para entregar, pero sí para practicar y consultar
- Finalizado para el Lunes 4/Sept/2023



# TP - Arbolado Porteño [Grupal]

Título de la columna	Tipo de dato	Descripción
long	Número flotante (float)	Coordenadas para geolocalización
lat	Número flotante (float)	Coordenadas para geolocalización
id_arbol	Número entero (integer)	Identificador único del árbol
altura_tot	Número entero (integer)	Altura del árbol (m)
diametro	Número entero (integer)	Diámetro del árbol (cm)
inclinacio	Número entero (integer)	Inclinación del árbol (grados)
id_especie	Número entero (integer)	Identificador de la especie
nombre_com	Texto (string)	Nombre común del árbol
nombre_cie	Texto (string)	Nombre científico del árbol
tipo_folla	Texto (string)	Tipo de follaje del árbol
espacio_ve	Texto (string)	Nombre del espacio verde
ubicacion	Texto (string)	Dirección del espacio verde
nombre_fam	Texto (string)	Nombre de la familia del árbol
nombre_gen	Texto (string)	Nombre del género del árbol
origen	Texto (string)	Origen del árbol
coord_x	Número flotante (float)	Coordenadas para localización
coord_y	Número flotante (float)	Coordenadas para localización



## Cierre



1. Repaso ejercicios
2. Python desde la terminal
3. Correr archivos .py
4. IDE - Spyder
5. Funciones
6. Diccionarios
7. Módulos
8. Manejo de archivos

## *Tareas para la próxima clase*



1. *Resolver Guía de Ejercicios de Python [Individual]*
2. *Resolver TP Arbolado [Grupal]*